



Angular Advanced RxJS and Operators



Peter Kassenaar –
info@kassenaar.com

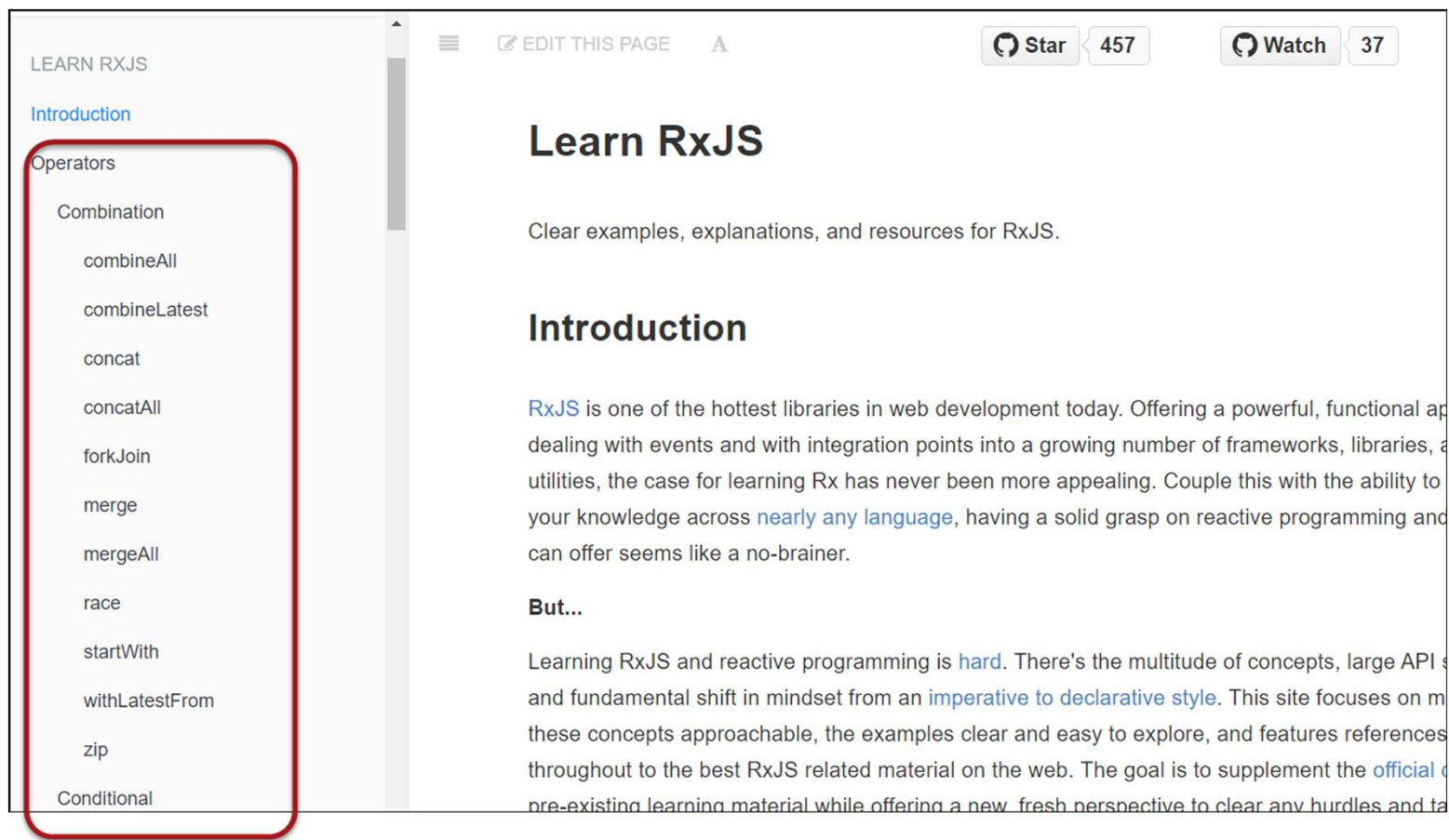


More on operators

Transforming your stream in the `.pipe()` function

The Problem with RxJS Operators...

- There are so many of them...



The screenshot shows the 'Learn RxJS' website. On the left, a sidebar lists various RxJS operators under the heading 'Operators'. The operators listed are: combineAll, combineLatest, concat, concatAll, forkJoin, merge, mergeAll, race, startWith, withLatestFrom, and zip. These operators are grouped under a 'Combination' sub-header. The main content area on the right is titled 'Learn RxJS' and 'Introduction'. It contains text about RxJS being a popular library and the challenges of learning it. The sidebar is highlighted with a red border, emphasizing the large number of operators.

LEARN RXJS

Introduction

Operators

Combination

- combineAll
- combineLatest
- concat
- concatAll
- forkJoin
- merge
- mergeAll
- race
- startWith
- withLatestFrom
- zip

Conditional

EDIT THIS PAGE

Star 457

Watch 37

Learn RxJS

Clear examples, explanations, and resources for RxJS.

Introduction

[RxJS](#) is one of the hottest libraries in web development today. Offering a powerful, functional approach to dealing with events and with integration points into a growing number of frameworks, libraries, and utilities, the case for learning Rx has never been more appealing. Couple this with the ability to transfer your knowledge across [nearly any language](#), having a solid grasp on reactive programming and the benefits it can offer seems like a no-brainer.

But...

Learning RxJS and reactive programming is [hard](#). There's the multitude of concepts, large APIs, and a fundamental shift in mindset from an [imperative to declarative style](#). This site focuses on making these concepts approachable, the examples clear and easy to explore, and features references throughout to the best RxJS related material on the web. The goal is to supplement the [official documentation](#) and pre-existing learning material while offering a new, fresh perspective to clear any hurdles and to

The image is a screenshot of the 'Learn RxJS' website. On the left, a sidebar contains a navigation menu. The 'Operators' section is highlighted with a red rounded rectangle. Below it, the 'Combination' sub-section is visible, listing several operators: `combineAll`, `combineLatest`, `concat`, `concatAll`, `forkJoin`, `merge`, `mergeAll`, `race`, `startWith`, `withLatestFrom`, and `zip`. The main content area on the right has a header 'Learn RxJS' with a subtitle 'Clear examples, explanations, and resources for RxJS.' Below this is an 'Introduction' section. The text in the introduction states that RxJS is a popular library for web development, but learning it is difficult due to its many concepts and the shift from imperative to declarative programming. A white text box is overlaid on the 'But...' section, containing the text: 'Learning RxJS and reactive programming is hard. There's the multitude of concepts, large API surface, and fundamental shift in mindset from an imperative to declarative programming. This site focuses on making these concepts approachable, the examples clear and easy to explore, and features references throughout to the best RxJS related material on the web. The goal is to supplement the official documentation and pre-existing learning material while offering a new, fresh perspective to clear any hurdles and take you to the next level of understanding.' At the top right of the page, there are buttons for 'Star' (457) and 'Watch' (37).

LEARN RXJS

Introduction

Operators

Combination

- `combineAll`
- `combineLatest`
- `concat`
- `concatAll`
- `forkJoin`
- `merge`
- `mergeAll`
- `race`
- `startWith`
- `withLatestFrom`
- `zip`

Conditional

Learn RxJS

Clear examples, explanations, and resources for RxJS.

Introduction

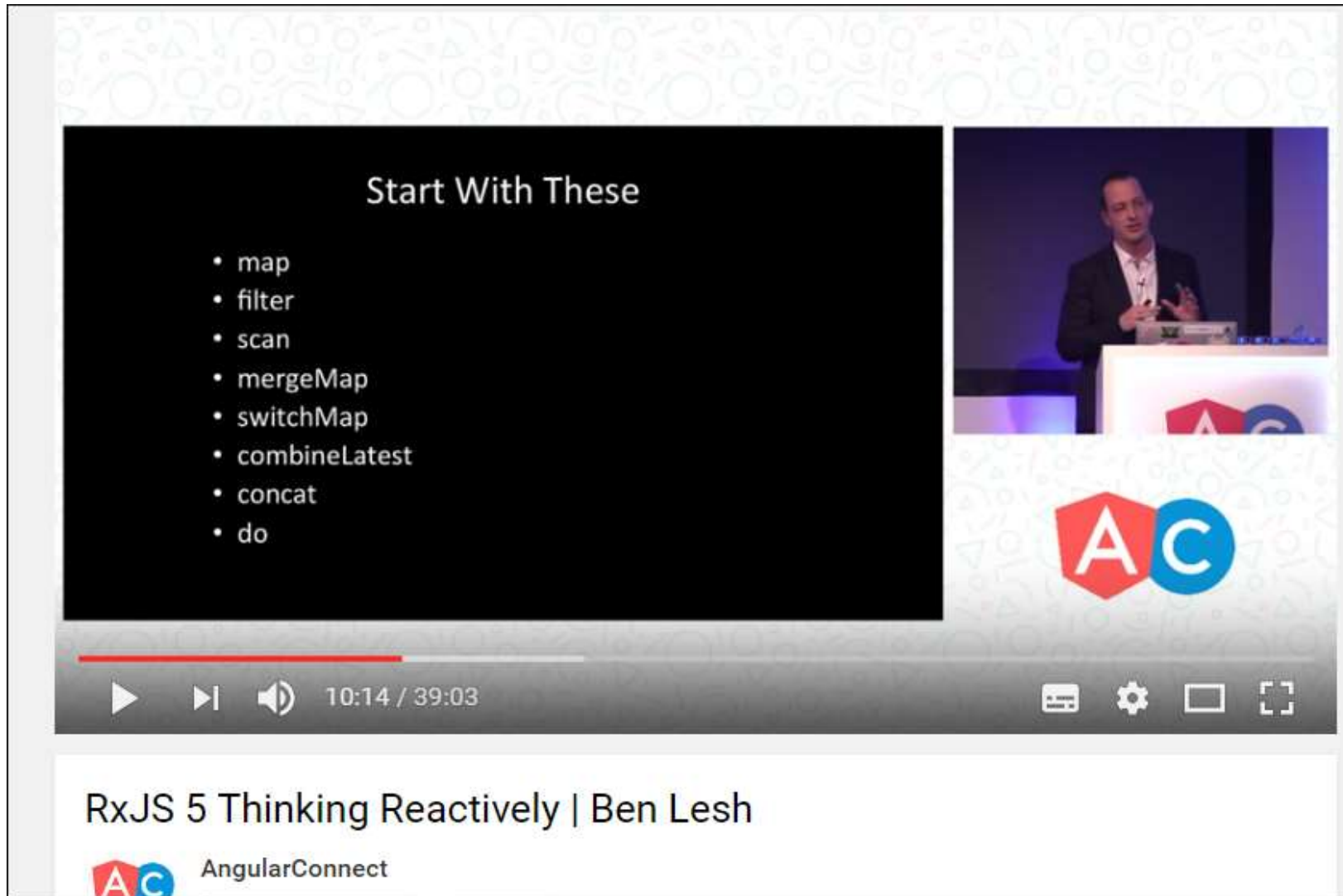
RxJS is one of the hottest libraries in web development today. Offering a powerful, functional approach to dealing with events and with integration points into a growing number of frameworks, libraries, and utilities, the case for learning Rx has never been more appealing. Couple this with the ability to transfer your knowledge across *nearly any language*, having a solid grasp on reactive programming and how it can offer seems like a no-brainer.

But...

Learning RxJS and reactive programming is *hard*. There's the multitude of concepts, large API surface, and fundamental shift in mindset from an *imperative to declarative* programming. This site focuses on making these concepts approachable, the examples clear and easy to explore, and features references throughout to the best RxJS related material on the web. The goal is to supplement the *official documentation* and pre-existing learning material while offering a new, fresh perspective to clear any hurdles and take you to the next level of understanding.

Star 457 Watch 37

<https://www.learnrxjs.io/>



The image shows a YouTube video player interface. The main video area displays a presentation slide titled "Start With These" with a list of RxJS operators: map, filter, scan, mergeMap, switchMap, combineLatest, concat, and do. To the right of the slide is a small inset video of the speaker, Ben Lesh, at a podium. The video player includes a progress bar at 10:14 / 39:03 and standard playback controls. Below the video, the title "RxJS 5 Thinking Reactively | Ben Lesh" and the "AngularConnect" channel logo are visible.

Start With These

- map
- filter
- scan
- mergeMap
- switchMap
- combineLatest
- concat
- do

10:14 / 39:03

RxJS 5 Thinking Reactively | Ben Lesh

AngularConnect

<https://www.youtube.com/watch?v=3LKMwkuK0ZE>

"Don't try to 'Rx everything'. Start with the operators you know, and go imperatively from there. There's nothing wrong with that."

- Ben Lesh

Example code

The screenshot shows the GitHub interface for the repository **PeterKassenaar / ng-rxjs-operators**. The repository name is circled in red. The page displays the repository's description, statistics, and a list of files.

Repository: PeterKassenaar / ng-rxjs-operators

Statistics: 3 commits, 1 branch, 0 releases, 1 contributor

Files:

File	Description	Time
src	Added first version of code.	22 hours ago
.angular-cli.json	Added first version of code.	22 hours ago
.editorconfig	Added first version of code.	22 hours ago
.gitignore	Updated .gitignore. Added yarn support	22 hours ago

<https://github.com/PeterKassenaar/ng-rxjs-operators>

This presentation - 2 sections

1. Basic Streams

- Create a stream, or multiple streams from scratch, based on a UI-element
- Subscribe to that stream(s) and do something

2. Operators

Demo the purpose and inner workings of some often used operators

Helpful resource - rxMarbles

RxJS Marbles

[sequenceEqual](#)

COMBINATION OPERATORS

[combineLatest](#)

[concat](#)

[merge](#)

[race](#)

[startWith](#)

[withLatestFrom](#)

[zip](#)

FILTERING OPERATORS

[debounceTime](#)

[debounce](#)

[distinct](#)

[distinctUntilChanged](#)

[elementAt](#)

[filter](#)

[find](#)

[findIndex](#)

[first](#)

[ignoreElements](#)

[last](#)

Interactive diagrams of Rx Observables

`combineLatest((x, y) => "" + x + y)`

Fork me on GitHub

Built on RxJS v5.0.3

<https://rxmarbles.com/>

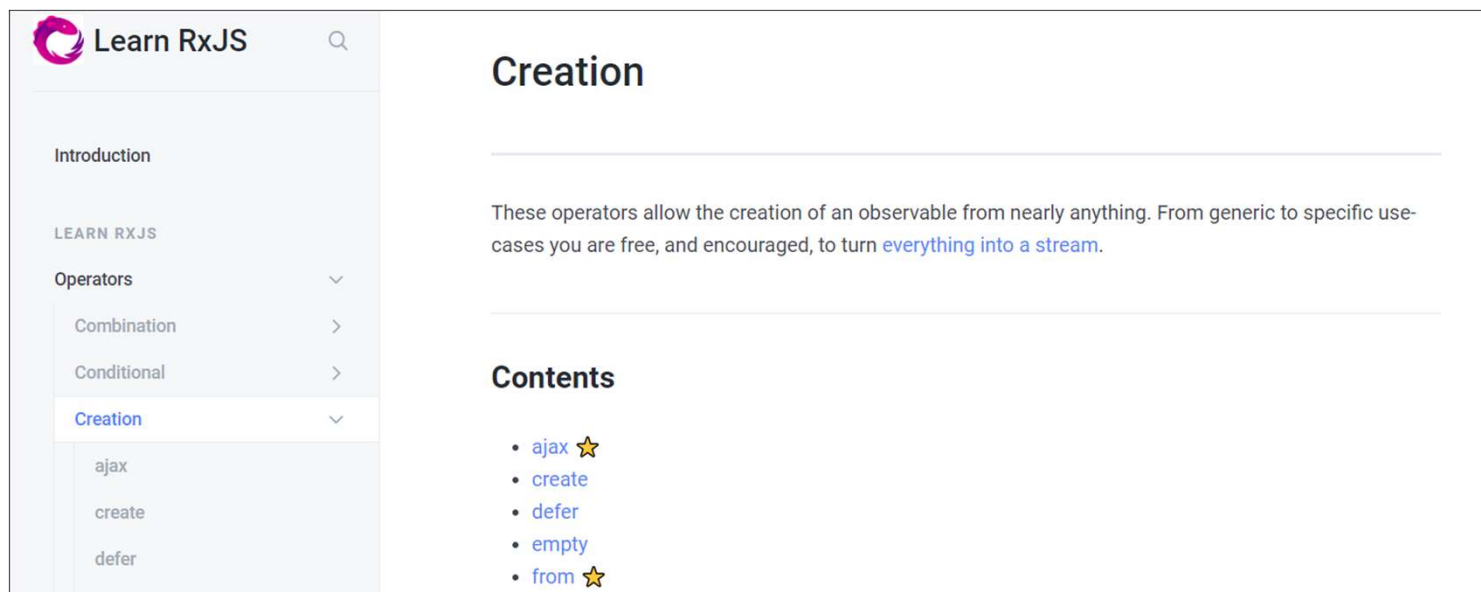


Basic streams

Creating observables from scratch

Turn something into an observable

- Basic creation operators
 - `create()` – create an observable manually
 - `from()` – turn an array, promise or iterable into an observable
 - `fromEvent()` – turn an event into an observable
 - `of()` – turn a variable into an observable, emit it's value('s) and complete
- <https://www.learnrxjs.io/learn-rxjs/operators/creation>



Creating an observable from a textbox

```
<input class="form-control-lg" type="text" #text1
      id="text1" placeholder="Text as a stream">
<p>{{ textStream1 }}</p>
```

```
textStream1 = 'Type some text above...';
@ViewChild('text1', {static: true}) text1; // two parameters for @ViewChild()

// Suggestion: break the onInit() up in smaller functions
ngOnInit(): void {
  this.onTextStream1();
}

onTextStream1() {
  fromEvent(this.text1.nativeElement, 'keyup')
    .subscribe((event: any) => this.textStream1 = event.target.value);
}
```


Result

Basic stream: we subscribe to chars coming from the textbox:

Text as a stream

Type some text above...

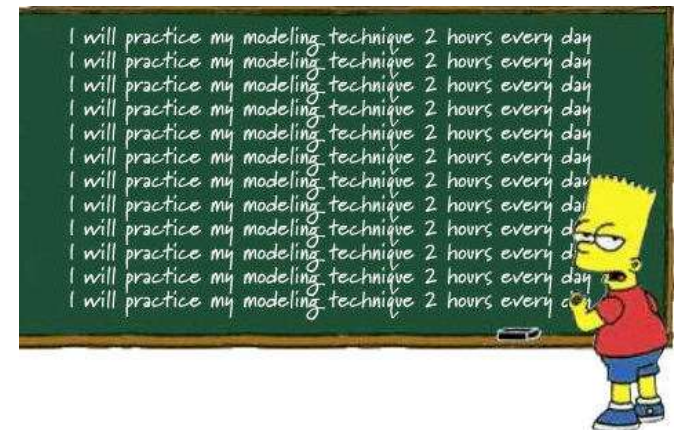
Basic stream: we subscribe to chars coming from the textbox:

 This is a stream...

This is a stream...

Workshop

- Create a textbox.
 - Add items that are typed in, to a Todo-list
 - Use an observable to subscribe to keyup-events.
- Capture key events, test if the key pressed is the Enter-key
 - If it's Enter, Add the current value of the textbox to a static array `todoItems[]`
- Example `../basic-stream/basic-stream.component.ts`
 - Start from scratch or expand this component



Using the pipe()

- We can transform the stream by adding operators to the pipeline
- Inside the pipeline we calculate new values and bind them to the UI, using generic attribute binding [...]

```
<button #btnRight class="btn btn-secondary">Right</button>
Mario is moved by observable streams from the button click.
<div>
  
</div>
```

Inside our class

```
// mario
position: any;
@ViewChild('btnRight', {static: true}) btnRight;
```

```
fromEvent(this.btnRight.nativeElement, 'click')
  .pipe(
    map(event => 10), // 1. map the event to a useful value, in this case 10px
    startWith({x: 100, y: 100}), // 2. start with an object of { 100, 100}.
    scan((acc: any, current: number) => { // 3, use the scan operator as reducer function.
      return {
        x: acc.x + current,
        y: acc.y
      };
    })
  )
  .subscribe(result => {
    this.position = result;
  });
```

The result of the previous operator is the input of the next operator in the pipeline

Result

Right

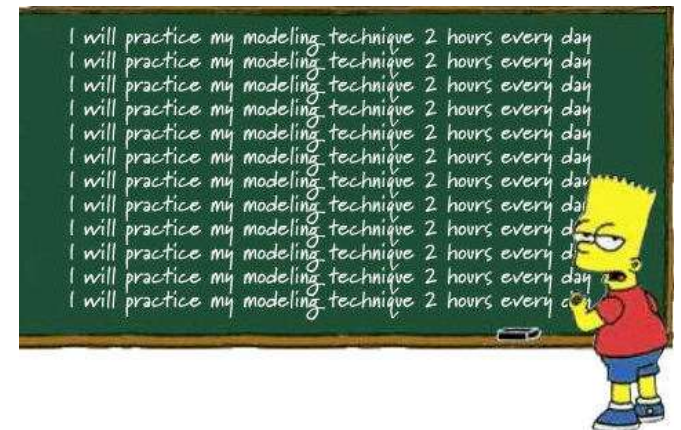
Mario is moved by observable streams from the button click.



Workshop

- Create button that moves Mario to the left.
- Example `../basic-stream/basic-stream.component.ts`
 - Start from scratch or expand this component

<https://github.com/PeterKassenaar/ng-rxjs-operators>



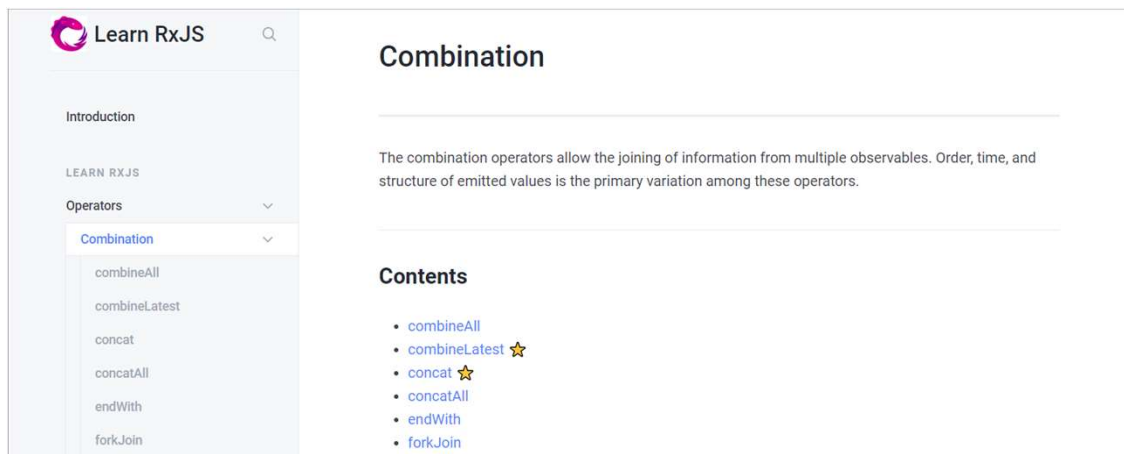


Combining streams

There are **a lot** of options to combine multiple streams into one stream

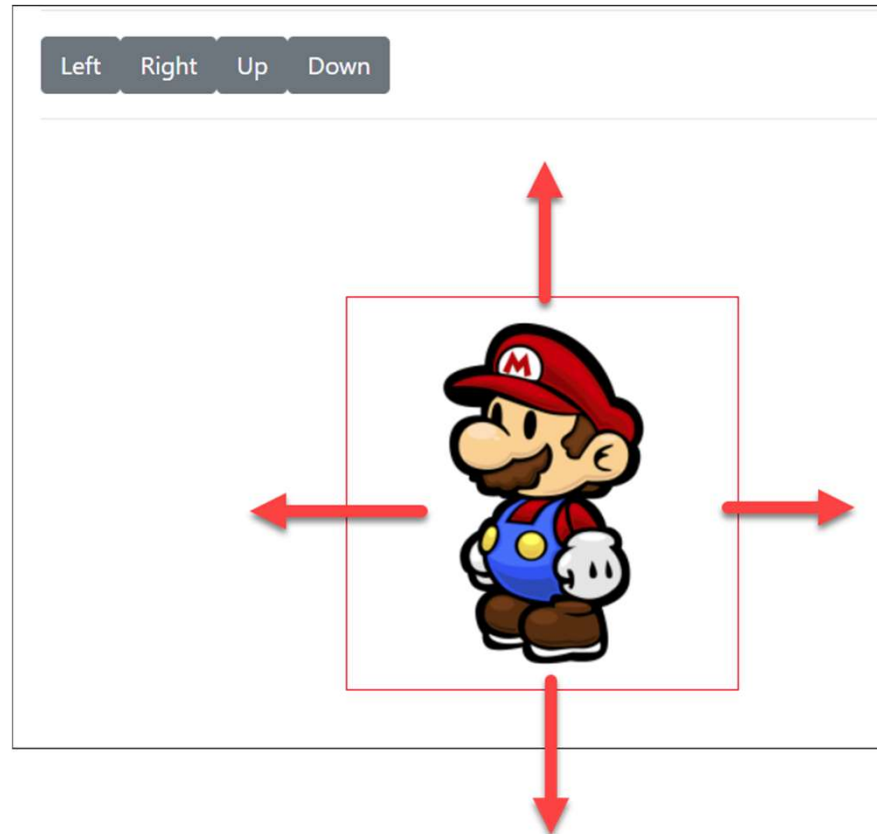
Options for combining streams

- `combineLatest()` - When any observable emits a value, emit the last emitted value from each
- `concat()` - Subscribe to observables in order as previous completes
- `forkJoin()` - When all observables complete, emit the last emitted value from each
- `merge()` - Turn multiple observables into a single observable
- `startWith()` - Emit given value first
- <https://www.learnrxjs.io/learn-rxjs/operators/combo>



Use case: move Mario

- Create four buttons to move Mario in different directions
 - all combined in one stream and one subscription



Template

```
<button #btnLeft> Left </button>
<button #btnRight> Right </button>
<button #btnUp> Up </button>
<button #btnDown> Down </button>
<hr>


```

```

position: any;
@ViewChild('btnLeft', {static: true}) btnLeft;
@ViewChild('btnRight', {static: true}) btnRight;
@ViewChild('btnUp', {static: true}) btnUp;
@ViewChild('btnDown', {static: true}) btnDown;

ngOnInit(): void {
  // Create multiple streams
  const right$ = fromEvent(this.btnRight.nativeElement, 'click')
    .pipe(
      map(event => {
        return {direction: 'horizontal', value: 10};
      }) // 10 px to the right
    );
  const left$ = fromEvent(this.btnLeft.nativeElement, 'click')
    .pipe(
      map(event => {
        return {direction: 'horizontal', value: -10};
      }) // -10 px to the left
    );
  ...
  // combine our streams
  merge(right$, left$, up$, down$)
    .pipe(
      startWith({x: 200, y: 100}),
      scan((acc: any, current: any) => {
        return {
          x: acc.x + (current.direction === 'horizontal' ? current.value : 0),
          y: acc.y + (current.direction === 'vertical' ? current.value : 0)
        };
      })
    ).subscribe(result => {
      this.position = result;
    });
}

```

Create 4 different streams

Merge the streams

One subscriber



Sequencing streams

Use streams to start other streams so you can use the combined behavior

Use case: drag & drop

- We want to move Mario around with the mouse: **drag-n-drop**
- We compose different streams for that:
 - `down$`, when the mouse is pressed
 - `move$`, when the mouse is moved, return the current mouse position
 - `up$`, when the mouse is released
- Again, we have only one (1) subscription
 - **Subscribe** to the `down$`. This is the initiator
 - After the initial event, **switch** to the `move$`: using the `switchMap()` operator
 - But only **until** the `up$` event occurs!
 - Then complete the observable and release Mario
- This translates to the following code:

// correction factor for image and current page. Your mileage may vary!

const OFFSET_X = 180;

const OFFSET_Y = 280;

// 1. With Drag and drop, first you capture the mousedown event

const down\$ = fromEvent(document, 'mousedown');

Compose streams

// 2. What to do when the mouse moves

const move\$ = fromEvent(document, 'mousemove')

.pipe(

map((event: **any**) => {

return {x: event.pageX - OFFSET_X, y: event.pageY - OFFSET_Y}; *// OFFSET as correction factor*

})

);

// 3. Capture the mouseup event

const up\$ = fromEvent(document, 'mouseup');

// 4. extend the down\$ stream and subscribe

down\$

.pipe(

// IF the down\$-event happens, we are no longer interested in it. Instead,

// we switch the focus to the move\$ event.

// BUT: we are only interested in the move until the mouse is released again.

*// So we add *another* pipe and use the takeUntil() operator.*

switchMap(event => move\$.pipe(

takeUntil(up\$)

)),

startWith(**this**.position)

)

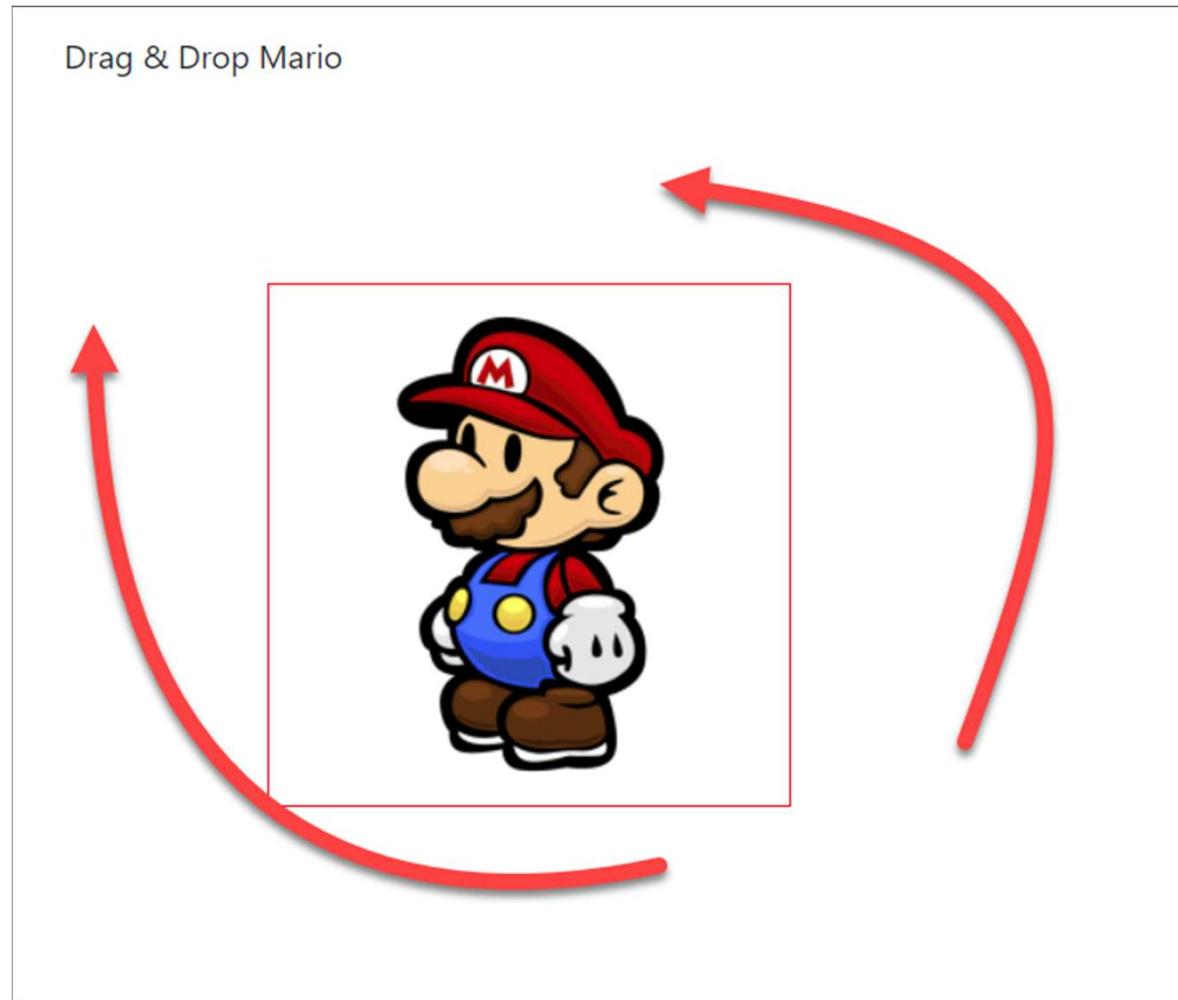
.subscribe(result => **this**.position = result);

**Switch execution to
another observable**

**Subscribe and
update position**

}

Result



`../streams/drag-drop-stream.component.html | ts`

An autocomplete/typeahead demo

An autocomplete/typeahead demo

bel



Belarus

Minsk



Belgium

Brussels



Belize

Belmopan



Palau

Ngerulmud

Template

As per usual – little HTML

```
<h4>An autocomplete/typeahead demo</h4>
<!--Get the keyword, the class is subscribed to this inputbox-->
<input type="text" placeholder="Country name..."
      #typeahead class="form-control-lg">
<hr/>
<!--Results-->
<ul class="list-group">
  <li class="list-group-item" *ngFor="let country of countries$ | async ">
    
    <p>
      <strong>{{ country.name }}</strong><br>
      {{ country.capital }}
    </p>
  </li>
</ul>
```

This time: using the
async pipe

Code

```
interface ICountry {  
  name: string;  
  capital: string;  
  flag: string;  
}  
  
const errorCountry: ICountry = {  
  name: 'Error',  
  capital: 'Not found',  
  flag: ''  
};
```

Creating interface and simple error message

```
@ViewChild('typeahead', {static: true}) typeahead;  
countries$: Observable<ICountry[]>;  
  
constructor(private http: HttpClient) {  
}
```

Inside the component class

```
ngOnInit(): void {  
  this.countries$ = fromEvent(this.typeahead.nativeElement, 'keyup')  
    .pipe(  
      filter((e: any) => e.target.value.length >= 2),  
      debounceTime(400),  
      map((e: any) => e.target.value),  
      distinctUntilChanged(),  
      switchMap(keyword => this.getCountries(keyword))  
    );  
}
```

Main method. See example code for comments

../streams/typeahead-stream.component.html | ts

Fetching the countries

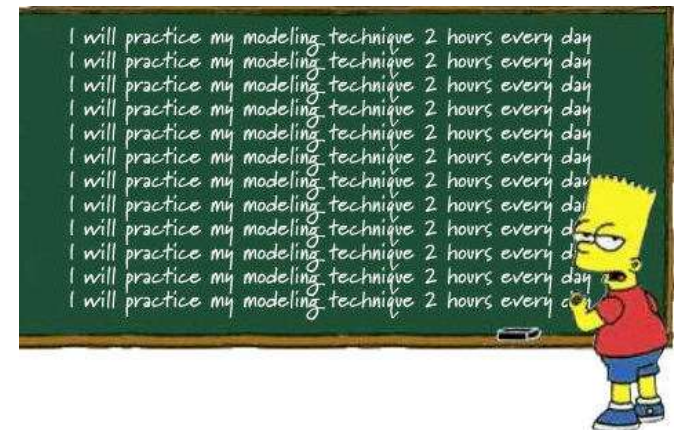
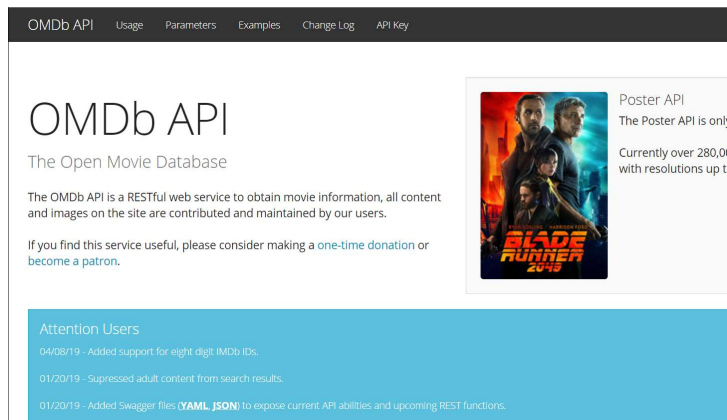
```
getCountries(keyword): Observable<ICountry[]> {  
  // 7. Create the actual http-call.  
  return this.http  
    .get<ICountry[]>(`https://restcountries.eu/rest/v2/name/${keyword}?fields=name;capital;flag`)  
    .pipe(  
      // 8. catch http-errors and return a 'not found' country  
      catchError(err => {  
        console.log(err);  
        return of([errorCountry]);  
      })  
    );  
}
```

Documentation:

- filter: <https://www.learnrxjs.io/learn-rxjs/operators/filtering/filter>
- debounceTime: <https://www.learnrxjs.io/learn-rxjs/operators/filtering/debouncetime>
- distinctUntilChanged: <https://www.learnrxjs.io/learn-rxjs/operators/filtering/distinctuntilchanged>
- switchMap: <https://www.learnrxjs.io/learn-rxjs/operators/transformation/switchmap>
- catchError: https://www.learnrxjs.io/learn-rxjs/operators/error_handling/catch

Workshop

- Search movies in the Open Movie Database.
 - `https://www.omdbapi.com/?apikey=f1f56c8e&s=[keyword]`
 - Create an AutoComplete inputfield for movie titels.
 - For instance, if you type 'ava...' it should show movies like Avatar, and more
- Example `../basic-stream/typeahead.component.ts`
 - Start from scratch or expand this component





More operators

Getting familiar with some of the more used operators

map() and mapTo()

```
const source = from([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);

// .map() - apply a function every emitted output.
source.pipe(
  map((val: number) => val = val * 10)
)
  .subscribe(result => this.mapData.push(result));

// .mapTo() - map the emission to a constant value
source.pipe(
  mapTo('Hello World')
)
  .subscribe(result => this.mapToData.push(result));
```

filter()

```
const source      = Observable.from([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);
const sourceCities = Observable.from([
  'Haarlem',
  'Breda',
  'Amsterdam',
  'Groningen',
  'Hengelo',
  ...
]);
```

*// filter() - only emit values that pass the provided condition.
// In this case: only even numbers are passed through.*

```
source.pipe(
  filter(val => val % 2 === 0)
)
.subscribe(result => this.filterData.push(result));
```

// Emit only the cities that starts with an 'H'.

```
sourceCities.pipe(filter(city => city.startsWith('H')))
.subscribe(result => this.cityData.push(result));
```

scan()

```
const source = Observable.from([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);
```

*// .scan() acts as the classic .reduce() function on array's. It takes an
// accumulator and the current value. The accumulator is persisted over time.*

```
source
  .pipe(
    startWith(0),
    scan((acc, curr) => acc + curr)
  )
  .subscribe(result => this.scanData.push(result));
```

concat()

- Concat subscribes to observables *in order*.
- “Only when the first one completes, let me know.”
- Then move on to the next one.
- Use `concat()` when the order of your observables matters.
- Example code: simulated delay.
 - The second observable (which is finished first) only emits when the first observable completes.

app/shared/services/data.service.ts

// constants that are used as pointers to some json-data

const BOOKS: string = 'assets/data/books.json';

const AUTHORS: string = 'assets/data/authors.json';

```
getConcatData(): Observable<any> {  
  // First call. Simulate delay of 1 second  
  const authors = this.http.get(AUTHORS)  
    .pipe(  
      delay(2000)  
    );  
  // Second call. Simulate delay of 2 seconds  
  const books = this.http.get(BOOKS)  
    .pipe(  
      delay(1000)  
    );  
  
  // return the concatenated observable. It will always deliver  
  // FIRST the results of the first call. No matter how long the delay.  
  return concat(authors, books);  
}
```

merge()

- Merge combines multiple observables into one single observable.
- It emits *as soon as* it gets a result.
- Use `merge()` when order of observables is not important.

```
getMergeData(): Observable<any> {  
  // First call. Simulate delay of 3 seconds, so this observable will emit last.  
  const authors = this.http.get(AUTHORS)  
    .pipe(  
      delay(3000)  
    );  
  
  // Second call. Simulate delay of 1 seconds, so this observable will emit first.  
  const books = this.http.get(BOOKS)  
    .pipe(  
      delay(1000)  
    );  
  
  // return the merged observable. BOOKS will be delivered first  
  return merge(authors, books);  
}
```



mergeMap()

- Merges the value of an inner observable into an outer observable.
- Need only the *last* value emitted? Use `.switchMap()`.
- Use this for example to retrieve results in a second observable, based on the output of a first observable

.mergeMap() result

See </app/shared/services/data.service.ts> for example code.

We are looking for books by authorName here. But we only have an authorId, not a name.



```
[
  {
    "title": "Web Development Library - TypeScript",
    "author": "Peter Kassenaar",
    "bookID": 1
  },
  {
    "title": "Web Development Library - Angular",
    "author": "Peter Kassenaar",
    "bookID": 2
  }
]
```

```

getMergeMapData(authorID: number = 0): Observable<any> {
  // first http-call, outer observable.
  return this.http.get(AUTHORS)
    .pipe(
      tap(response => console.log(response)),
      map((authors: any[]) => {
        console.log(authors);
        // find the correct author, using the array .find() method
        return authors.find((author: any) => author.id === authorID);
      }),
      mergeMap((author: any) => {
        if (author) {
          // second http-call, inner observable
          return this.http.get(BOOKS)
            .pipe(map((books: any[]) => {
              // filter books, bases on authername we found earlier.
              return books.filter((book: any) => book.author === author.name);
            }));
        } else {
          // nothing found. Return empty array
          return of([]);
        }
      })
    );
}

```

.forkJoin()

- Make multiple (http) requests and return one combined response *once all observables are completed*.
- `.forkJoin()` returns an array with the last emitted value from each observable.
- Compose results as per your needs

```


getForkJoinData(): Observable<any> {
  return forkJoin(
    // first call
    this.http.get(AUTHORS)
      .pipe(
        delay(3000)
      ),
    // second call
    this.http.get(BOOKS)
  )
  .pipe(
    map((data: any[]) => {
      // data is now an array with 2 objects, b/c we did 2 http-calls.
      // First result, from the http-call to AUTHORS
      const author: any = data[0][0]; // Get just first author from file.
      // Second result, from the http-call to BOOKS
      const books: any[] = data[1];
      // Compose result, in this case adding the books to the extracted author.
      author.books = books.filter(book => book.author === author.name);
      return author;
    })
  );
}

```

Other interesting / often used Operators





- `from()`, `of()` - create observables from almost everything
- `fromEvent()` - create observable from a DOM-event.
- `debounce()`, `debounceTime()` - Discard emitted values that take less than the specified time between output
- `tap()` - utility operator - transform side-effects, such as logging
- ...and much more... See <http://www.learnrxjs.io>
- See also <https://rxjs-dev.firebaseapp.com/>

ARTICLES SPEAKING TRAINING WORKSHOPS VIDEOS




My name is [Cory Rylan](#). [Google Developer Expert](#) and Front End Developer at [VMware Clarity](#). [Angular Boot Camp](#) instructor.

[Follow @coryrylan](#)



Angular Multiple HTTP Requests with RxJS




Cory Rylan
Nov 15, 2016
Updated Jun 18, 2019 - 5 min read

angular rxjs

This article has been updated to the latest version of [Angular](#) 9 and tested with Angular 8. The content is likely be applicable for older Angular 2 or other previous versions.

A typical pattern we run into with single page apps is to gather up data from multiple API endpoints and then display the gathered data to the user. Fetching numerous asynchronous requests and managing them can be tricky but with the Angular's Http service and a little help from the included RxJS library, it can be accomplished in just a few of lines of code. There are multiple ways to handle multiple requests; they can be sequential or in parallel. In this post, we will cover both.





Angular Form Essentials

Learn the essentials to get started creating amazing forms with Angular!


[GET E-BOOK NOW!](#)

<https://coryrylan.com/blog/angular-multiple-http-requests-with-rxjs>

More information




Sign in / Sign up

**Netanel Basal** [Follow](#)
Jan 24 · 3 min read

RxJS—Six Operators That you Must Know

```
class TakeSubscriber<T> extends Subscriber<T> {  
  private count: number = 0;  
  
  constructor(destination: Subscriber<T>, private total: number) {  
    super(destination);  
  }  
  
  protected _next(value: T): void {  
    const total = this.total;  
    const count = ++this.count;  
    if (count <= total) {  
      this.destination.next(value);  
    }  
  }  
}
```

 Never miss a story from **NetanelBasal**, when you sign up for Medium. [Learn more](#)

GET UPDATES

<https://netbasal.com/rxjs-six-operators-that-you-must-know-5ed3b6e238a0#.ocz4xfwkl>



Daniele Ghidoli

About me

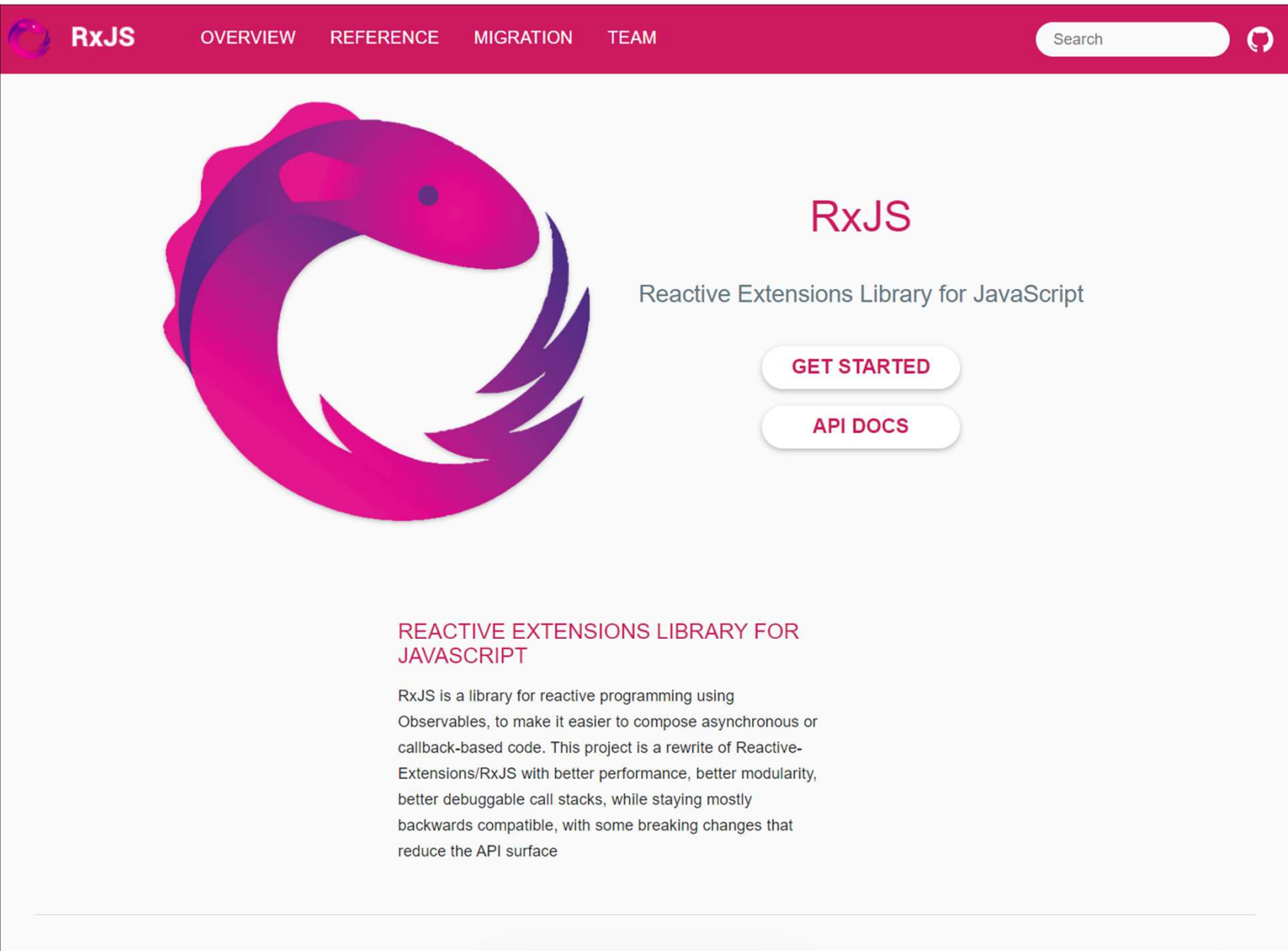


October 22, 2016

Combining multiple Http streams with RxJS Observables in Angular2



<http://blog.danieleghidoli.it/2016/10/22/http-rxjs-observables-angular/>



<https://rxjs-dev.firebaseio.com/>

[reactive.how](#)
2.0-alpha.4

[Launchpad for RxJS →](#)

[Follow](#)
[Stay tuned](#)

ESSENTIALS	REDUCING	TAKING	SKIPPING	COMBINING	TIME	CONCATENATING
map	reduce	take	skip (soon)	combineLatest	delay	startWith (soon)
filter	max	takeWhile	skipWhile	zip	debounceTime	endWith (soon)
merge	count	first	takeLast		throttleTime	concat (soon)
scan			last			

Learn RxJS operators and Reactive Programming principles

Launchpad for RxJS

debounceTime vs throttleTime

Decks of cards for RxJS: learn, compare and memorize

scan

reduce

map

filter

zip

combineLatest

<https://reactive.how/>

Workshop

- Create a call to the Open Movie Database API, using a keyword to search for 10 movies
 - <http://www.omdbapi.com/?apikey=f1f56c8e&s=<keyword>>
- Create an additional call to get details for every movie returned. Use the `imdbID` property for that
 - <http://www.omdbapi.com/?apikey=f1f56c8e&i=<imdbID>>
- Display results in the UI, first a list of movies, then details per movie as soon as they come available.
- What is your best solution? Multiple ways of doing this!

